(12) UK Patent Application (19) GB (11) 2 373 888 (13) A

(43) Date of A Publication 02.10.2002

(21) Application No 0108096.9

(22) Date of Filing 30.03.2001

(71) Applicant(s)
Toshiba Research Europe Limited
(Incorporated in the United Kingdom)
32 Queens Square, BRISTOL, BS1 4ND, United Kingdom

(72) Inventor(s)
Craig Dolwin

(74) Agent and/or Address for Service
Marks & Clerk
57-60 Lincoln's Inn Fields, LONDON, WC2A 3LS, United Kingdom

(51) INT CL^7
G06F 11/36

(52) UK CL (Edition T)
G4A AFMP

(56) Documents Cited
GB 2330428 A
EP 0458559 A2
US 6158018 A
US 6128751 A
US 6049672 A
US 5938774 A

(58) Field of Search
UK CL (Edition S) G4A AEC AEF AFMP
INT CL^7 G06F 11/36
EPODOC, WPI, PAJ, INSPEC

(54) Abstract Title
Dynamic vector address allocation for a code patching scheme

(57) The present invention relates to a system for implementing patch code in a processor. As a processor progresses through a program, it issues a program address which is used to obtain the next instruction from a program ROM. However, if the program ROM contains bugs or needs to be updated then a patch is provided in a RAM. In the present invention, the addresses issued by the processor are monitored to see if they correspond to an address identified as starting a piece of code which needs to be patched. In these circumstances, the processor is controlled to branch to the corresponding patch code.

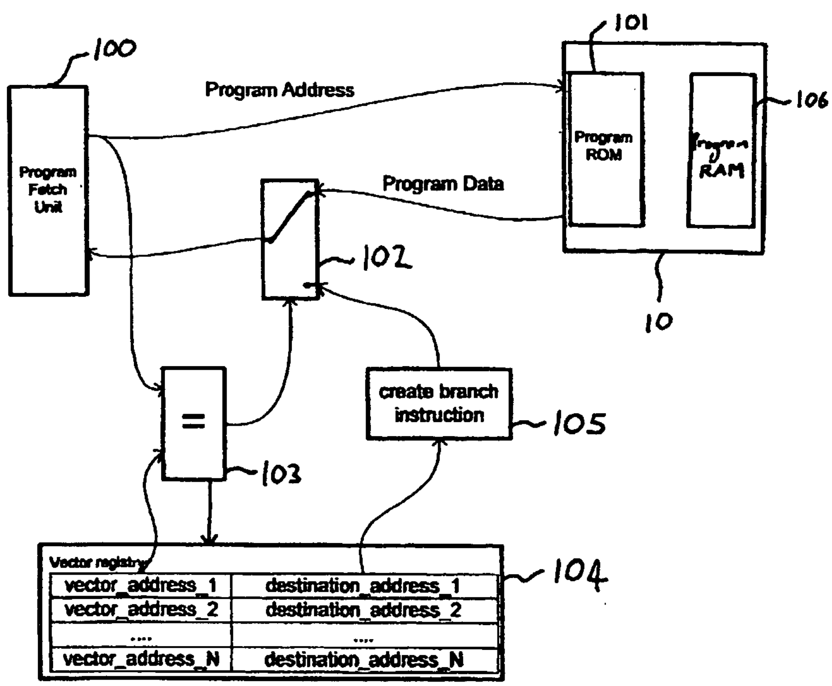

Figure 2

GB 2 373 888 A

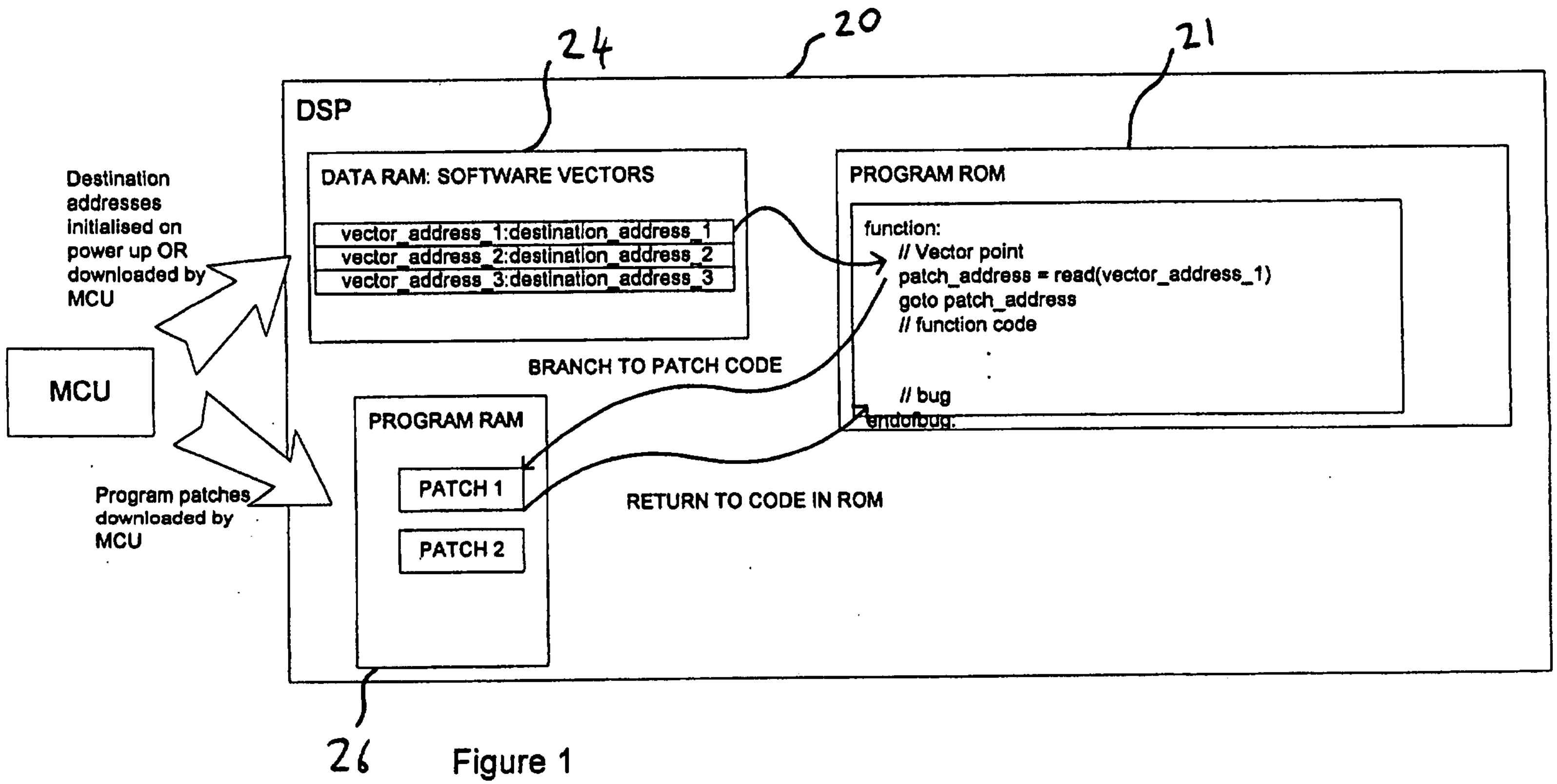
20
24
21
DSP
DATA RAM: SOFTWARE VECTORS
vector_address_1:destination_address_1
vector_address_2:destination_address_2
vector_address_3:destination_address_3
PROGRAM ROM
function:
// Vector point
patch_address = read(vector_address_1)
goto patch_address
// function code
// bug
endofbug:
BRANCH TO PATCH CODE
RETURN TO CODE IN ROM
PROGRAM RAM
PATCH 1
PATCH 2
MCU
Destination addresses initialised on power up OR downloaded by MCU
Program patches downloaded by MCU
26

Figure 1

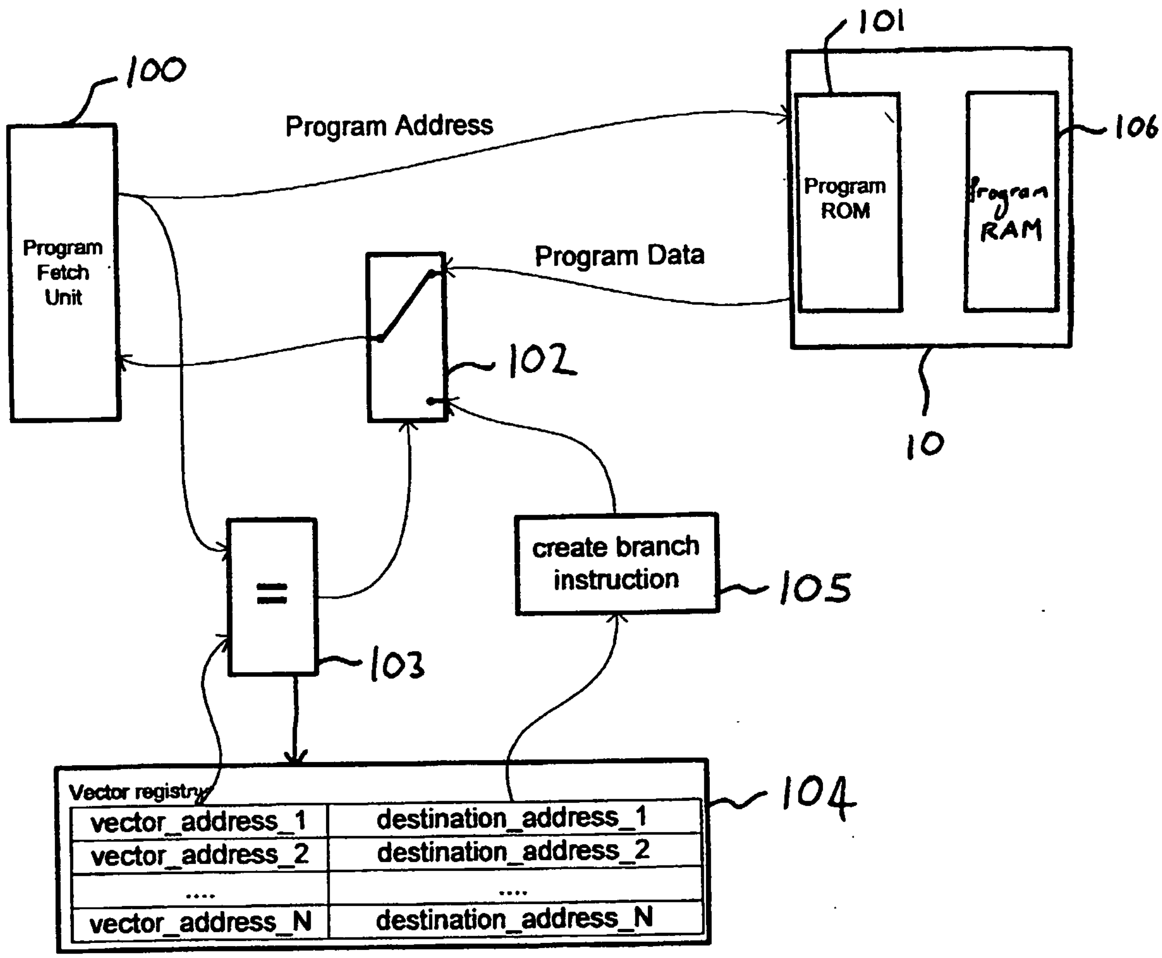
100
Program Fetch Unit
Program Address
101
Program ROM
Program RAM
106
10
Program Data
102
=
103
create branch instruction
105
Vector registry
vector_address_1
destination_address_1
vector_address_2
destination_address_2
....
....
vector_address_N
destination_address_N
104

Figure 2

2373888

DYNAMIC VECTOR ADDRESS ALLOCATION FOR A CODE PATCHING SCHEME

The present invention relates to a system for implementing patch code in an embedded processor.

An ever increasing number of devices are being produced which have computer processors integrated into them to perform some or all of the control of the device. These devices can be integrated into all sorts of different applications and allow cheaper, more rapid and more flexible development of products. These devices are also ever increasing in performance and thus allow more and more functions which would previously have been carried out using bespoke hardware to be carried out using these kinds of processor systems. One particular application where these devices provide significant advantage is in the field of digital signal processing. Digital signal processors can now be developed for use in a variety of applications with the flexibility that the controlling software can be produced and tested initially without having to develop expensive prototype hardware.

The controlling software is developed and then stored in a ROM in an embedded processor system, such as a DSP. However, due to commercial pressure and long lead times, the ROM masks used for an embedded DSP system have to be finalised before the rest of the software and system have been completed. This means the code used in the DSP is often not be thoroughly tested before being laid down in the ROM mask . It is therefore quite likely that bugs will be found following the production of the masked DSP. To address this problem (without replacing the ROM with expensive and unrepresentative alternatives such as FLASH memory or RAM), a quantity of program RAM is allocated for use as patch memory. Patch memory provides a convenient way of providing updated code to replace code on the ROM which contains bugs or is otherwise obsolete.

In a basic patch memory system the program code in ROM contains a number of 'vector points' at which point the address of the next program address is read from data RAM. This allows the microprocessor control unit (MCU) to download into the DSP's data RAM a new set of program vectors. In addition, the MCU also downloads into program RAM the new patch code. The new program vectors will point to the address of the new program code in program RAM. The new program includes all the code from the vector point up to and including the bug fix. Once the patch code is executed, it will then branch back to the ROM code.

Figure 1 shows an example of a prior art DSP system 20. The program ROM 21 contains a series of instruction codes. Execution of the program codes continues until it reaches a certain point, shown in Figure 1 indicated as "//Vector point". At this point the vector address 1 is referred to in the data RAM 24. If the next piece of code is satisfactory and does not require patching then the destination address corresponding to the vector address simply points to the next section of code in the program ROM. However, as in the example shown in Figure 1, where the subsequent code incorporates a bug ("//bug") then the destination address corresponding to the vector address is modified in the data RAM point to the appropriate piece of patch code in the program RAM 26. Execution of code then continues through the patch code whereupon the system reverts to obtaining instructions from the program ROM at a point beyond the bug.

Since only a limited number of vector points can be used in this way, the size of any patch is dependant on the distance of the bug from the vector point rather than the size of the changed code. This makes this method of patching code very inefficient and hence requires a large amount of patch RAM for a given number of bugs.

Therefore according to the present invention there is provided an instruction code control system for a processor comprising: a first memory for storing program instructions; a second memory for storing program instructions; a registry for storing a list of vector addresses and respective destination addresses; and a comparator for

comparing the vector addresses stored in the registry with addresses received from the processor, wherein when the comparator detects that an address from the processor corresponds to one of the vector addresses stored in the registry, a branch instruction along with the destination address corresponding to the respective vector address is sent to the processor.

The present invention also provides a method of controlling an instruction control system comprising: receiving address data from a processor; comparing said address data with one of more address vectors stored in a registry to determine there is a match; when no match is not identified, outputting an instruction code stored in a memory corresponding to the received address data and when a match is identified, outputting a branch instruction along with a destination address, the destination address being obtained from said registry according to the matched vector address.

This present invention provides a scheme for implementing patch code in an embedded processor system such as a DSP by using hardware to determine if faulty ROM program code is about to be executed. The processor is then redirected to execute a 'patch' of code which will preferably be located in program RAM. Associated with each vector address is a 'destination address'. The destination address is the value of the new address loaded into the program counter following the detection of the vector address. Ideally to reduce software overhead the destination addresses are preferably programmable and preferably located in hardware registers.

A hardware vector scheme removes the requirement for vector points to be hard coded into the ROM. A hardware vector can be implemented using several techniques but in principle it operates by monitoring the dispatch of program memory reads and when it detects a specific address ('vector address') it stops the normal flow of the processor and start executing alternative code defined by an associated 'destination address'. The vector addresses and destination addresses are both held in registries that can preferably be written to and read by the processor. This allows the vector address to be located as close as possible to the code containing a bug or needing replacement. In addition, it allows the possibility of implementing dynamic vector address allocation

To further increase the number of bugs that can be patched, a dynamic vector address scheme may also be provided it which the vector points according to which code is being executed at any particular time may be modified as a new section of code is to be executed. Dynamic vector address allocation will allow the number of vector addresses to exceed the number of hardware registers supplied.

Dynamic vector address allocation enhances the hardware vector scheme by re-using the vector registers when the processor is executing different functions. So initially a vector point is added to the beginning of a function. This code is then used to reload the vector registers using a unique set of vector and destination addresses for this particular function i.e. a set of bug fixes for the function. A further vector point may be added to the end of this function whereupon the default vector register set is reloaded. With a dynamic vector address allocation system the number of patches available within a system is equal to the number of vector registers multiplied by the number of occasions the vector register set is reloaded. Therefore, by reloading the vector register set more often, the number of patches available within the system can be increased, as required.

By placing some patch code at the beginning of the major functions this patch code can be used to dynamically change the vector and destination addresses to fix bugs specific to that function. With this technique the number of bugs that can be fixed with a given number of vector registers can be increased.

The patch code located at the beginning of a function can be implemented using either a software vector or a hardware vector. Using a hardware vector would involve including a vector register in the list of vector registers which corresponds to the end of the current function and causes the processor to branch to a subroutine (e.g. a patch in the program RAM) to load the next set of vector registers, or to load the default registers. The use of software registers relates to the software designer implementing the dynamic vector allocation scheme in software. This involves anticipating the possibility of the need for patches when the original software is developed. In this way, when the

original software is written to be stored on the ROM, at the start of a function or section of code, the processor would initialise the vector register set with a new group of patch vectors, specific to the function about to be executed. By using software vectors at that stage there is no need to include vector registers in list to deal with updating the registry and so all the hardware registers can be reserved for potential bug fixes within the function.

The system of the present invention can also be used advantageously for debugging code. In most embedded systems the hardware provides a function to allow the designer to stop the execution of the processor when it reaches a certain location in the program. This allows the designer to investigate (using software tools) the state of the processor. The location at which the program is halted is totally flexible and is determined by the designer. This facility is used during the testing of the software and the system and is called a "hardware breakpoint". The patch vector system described in this patent can also be used to implement a set of hardware breakpoints.

Hardware breakpoints could be used for adding patch code but extra breakpoints would be needed to avoid the problems of the prior art. Furthermore, most hardware breakpoints vector to the same location in memory and the software then determines which line of code caused the interrupt. (This can be done by the hardware storing the start and destination addresses of the last N program branches). However, the overhead of checking the source in a patch code system would be considerable. However, to use the proposed patch mechanism as a set of hardware breakpoints during debugging would avoid the need for traditional hardware breakpoints and allow simpler debugging.

The vector registry could be loaded with vector addresses corresponding to the desired breakpoints and then the program reaches those points, the operation can branch to a debugging or the like to allow the programmer to analyse the status of the processor. Thus, the only difference between the normal use of the machine and the debugging operation is the set of vector registers loaded.

The present invention can be used in the design of all types of embedded processor applications, but is of particular use in embedded DSP applications especially for wireless mobile systems.

Use of the dynamic vector address allocation scheme further enhances the hardware vector scheme by increasing the number of vectors to a very large value.

A specific embodiment of the present invention will now be described with reference to the drawings in which:-

Figure 1 shows a schematic representation of a prior art patch implementation scheme; and

Figure 2 shows a schematic representation of the system according to the present invention.

An embodiment of the present invention will now be described with reference to Figure 2. The construction shown comprises a program fetch unit 100 of a processor for extracting program instructions stored in one or more memories. These program instructions are principally stored in a ROM 101. However, in order to deal with the possibility of the need to update the program instructions stored in the program ROM, for example due to bugs in the code, some RAM is provided. A portion of this RAM, the program RAM 106, is loaded with additional program instruction codes which can be accessed by the processor. In this way, if a portion of the program stored in the ROM 101 contain errors or needs to be updated, then these programs can be provided in the program RAM 106.

In order to access the program instructions stored in the program memories, the processor controls the program fetch unit 100 to access an address in the program memories 101,106. An address comparator 103 is provided for monitoring the program addresses to determine whether an address relates to a part of the program which needs to be patched. The comparator 103 controls a switch 102 which determines whether program data is provided from the program memory 10 or from a branch generating unit

105 for generating branch instructions to redirect the operation of the device to obtain program instructions from a alternative location.

In operation, the program fetch unit 100 obtains program instructions from the program memory 10. The program fetch unit sends the address of the next instruction which is to be executed and passes this address to the program memory 10. Normally this will initially be a location in the program ROM 101. The program instructions stored at the program address specified by the program fetch unit 100 is extracted from the program ROM 101 and passed to the switching unit 102. Under normal operation, the switching unit simply passes the program instruction data received from the program memory 10 back to the program fetch unit 100 which then sends the program instruction on for execution by the processor.

The program address provided by the program fetch unit 100 is then updated to identify the address of the next instruction required by the processor. The program fetch unit then sends out the new address to the memory unit 10 which returns the appropriate program instruction to the program fetch unit via the switching unit 102 as described above. This process continues until the program fetch unit requests a program instruction code which has been determined to be invalid and therefore a patch needs to be used instead.

Every time the program fetch unit issues the program address of the next instruction which it requires, the program address is also sent to the comparator unit 103. This unit compares the program address with a plurality of vector addresses stored in a vector registry 104. If the program address issued by the program fetch unit 100 corresponds to one of the vector addresses stored in the vector registry, then the comparator unit 103 controls the switch unit 102 to switch over to receiving instructions from the branch instruction generator 105. The branch instruction generator then sends a branch instruction along with the destination address corresponding to the vector address in the vector registry 104. This is passed to the program fetch unit 100 via the switching unit 102. The program fetch unit then passes the branch instruction on to the processor to be executed as normal.

The processor is oblivious to the source of the branch instruction and simply executes it like any other instruction. The destination address associated with the branch instruction is then used to update the program address issued by the program fetch unit. In this way, when the program fetch unit next issues a request for a program instruction to the program memory 10, the new destination address is used. This would normally identify the piece of patch code which is loaded into the program RAM 106 of the program memory 10. Execution of the patch program code continues in this way until the end of the patch. The final instruction of the patch code would typically be a branch instruction back to a position in the original program in the program ROM 101 at a point beyond the portion of code which the patch has replaced.

The patch program may only be a single instruction or it may be one or more complete sub-routines. However, the patch only needs to be as long as the instructions originally stored on the ROM which need replacing. For example, if only a single instruction is incorrect in the original program stored in the ROM, then the patch only needs to replace that instruction. Consequently, the patch code will be very short (only requiring space for the replacement instruction and possibly one or two other instructions like the returning branch instruction) saving the scarce RAM memory. In contrast, in the prior art, it would have been necessary to replace all the instructions preceding the erroneous instruction back to the previous vector point.

Once the patch is completed and the program fetch unit is again obtaining instructions from the program ROM 101, then operation continues as before. The comparator unit 103 monitors the program address issued by the program fetch unit and if it identifies an address which corresponds to another vector address in the vector registry then the switch unit 102 is again switched over to receive a branch instruction and program instructions are then received from the program RAM. In this way, a large number of patches can be implemented simply by identifying the program address with the code in the program ROM to be replaced begins and the destination address where the replacement code is stored. This occupies relatively little of the available RAM leaving more of the RAM available for storage of programs or for more vectors. In addition,

because all of the sub-routine does not have to be patched, the program RAM can store considerably more patches and/or vectors.

However, in a complex system with potentially thousands of instructions, the number of patches may become very large and this will result in a large number of vector addresses and corresponding destination addresses having to be stored in the vector registry. Therefore, the present invention further provides a system in which the vector addresses stored in the vector registry are dynamically updated.

The processor is able to read and write to the registry and so can load a new set of vector registers into the vector registry for operation on each new section of program. For example, if the processor is intended to run a program which includes several different functions, then each time the processor switches from one function to the next function, it can update the vector registers in the vector registry with those needed for the program function about to be operated. Similarly, as the vector registers are updated, the patches stored in the program RAM can also be updated if necessary.

Thus as the processor reaches the end of a function of a predetermined piece of code, it then updates the vector registry with a new set of registers and possibly also a new set of patches are loaded into the program RAM. The processor then continues with the next instruction as before.

In order to update the vector registry at the end of a section of code so that when the next section of code is being executed, the appropriate registers are in the vector registry 104, then one of the registers can be arranged to detect when the program address corresponds to the end of the section of code causing a branch instruction to be issued. This directs operation of the processor to a section of code either in the RAM or in the ROM which arranges to update the vector registry ready for the next section of code.

Alternatively, at the end of a section of code, the processor could be controlled to update the registry to a default set of registers and as each new section of code begins, the registry is specifically updated with the registers for that section.

In an alternative arrangement, rather than use one of the vector registers to identify the end of a section of code and cause the vector registry to be updated, the code itself could include an instruction at the time the code is originally written which causes the processor to update the vector registry. Clearly at the time the program is written, the programmer does not know which parts of the code will ultimately need patching but he can still divide the program into manageable sections with the updating instruction provided between these sections. This avoids wasting a vector registry to do the updating task. Furthermore, if the sections of code chosen by the programmer are too large or inappropriate, for example because one section contains a large number of patches and so need to have the registry updated one or more within the section then

By changing the vector registry for different parts of a program, the potential number of patches which can be implemented is effectively unlimited and each separate piece of program or function can include its own set of vector registers to fix the bugs or provide the update for that piece.

CLAIMS

1. An instruction code control system for a processor comprising:

a first memory for storing program instructions;

a second memory for storing program instructions;

a registry for storing a list of vector addresses and respective destination addresses; and

a comparator for comparing the vector addresses stored in the registry with addresses received from the processor, wherein

when the comparator detects that an address from the processor corresponds to one of the vector addresses stored in the registry, a branch instruction along with the destination address corresponding to the respective vector address is sent to the processor.

2. An instruction code control system according to claim 1 wherein the registry is provided in part of the second memory.

3. An instruction code control system according to any one of the preceding claims, further comprising an output means for providing instruction codes from one of the first and second memory means when the address from the processor does not correspond to one of the vector addresses stored in the registry.

4. An instruction code control system according to any one of the preceding claims, wherein the first memory is a read only memory.

5. An instruction code control system according to any one of the preceding claims, wherein the contents of the registry are updatable in use.

6. An instruction code control system according to claim 5, wherein the system is adapted to load a new set of vector addresses into the registry.

7. An instruction code control system according to claim 6, wherein one of the vector addresses in the registry identifies an address to cause said new set of vector addresses to be loaded into the registry.

8. An instruction code control system according to claim 6, wherein said new set of vector addresses is loaded into the registry in response to a signal from the processor.

9. An instruction code control system according to any one of the preceding claims, wherein the second memory is a rewritable memory.

10. An instruction code control system according to claim 9, wherein the contents of the second memory are updatable in use.

11. An instruction code control system according to claim 10, wherein the system is adapted to load new program instructions into the second memory in use.

12. A method of controlling an instruction control system comprising:

receiving address data from a processor;

comparing said address data with one of more address vectors stored in a registry to determine there is a match;

when no match is not identified, outputting an instruction code stored in a memory corresponding to the received address data and

when a match is identified, outputting a branch instruction along with a destination address, the destination address being obtained from said registry according to the matched vector address.

13. A method according to claim 12, wherein said instruction code stored in the memory is stored in one of a read only memory and a rewritable memory.

14. A method according to claim 12 or 13, further comprising reloading vector addresses and corresponding destination addresses into said registry in response to an update instruction.

15. A method according to claim 14 further comprising adding a vector address to said registry to identify when to reload vector addresses and corresponding destination addresses into said registry.

16. A method of controlling an instruction control system substantially as described herein with reference to figure 2 of the accompanying drawings.

17. An instruction code control system substantially as described herein with reference to figure 2 of the accompanying drawings.

The Patent Office

INVESTOR IN PEOPLE

Application No:	GB 0108096.9	**Examiner:**	Ruth Atkinson
Claims searched:	All	**Date of search:**	12 December 2001

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.S): G4A AEC, AEF, AFMP

Int Cl (Ed.7): G06F 11/36

Other: Online: WPI, EPODOC, PAJ, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	GB 2330428 A	(WINBOND) See page 4 line 1 - page 6 line 6	1-6, 8-14
X	EP 0458559 A2	(SCHLUMBERGER) See column 1 line 40 - column 2 line 35	1-4, 9, 12, 13
X	US 6158018	(BERNASCONI) See 'Brief description of preferred embodiments'	1-6, 9-14
X	US 6128751	(YAMAMOTO) See column 2 line 31 - column 5 line 5	1-4, 9, 12, 13
X	US 6049672	(SHIELL) See column 12 line 58 - column 13 line 24	1-7, 9-15
X	US 5938774	(HSU) See 'Summary of invention'	1-6, 9-13

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		